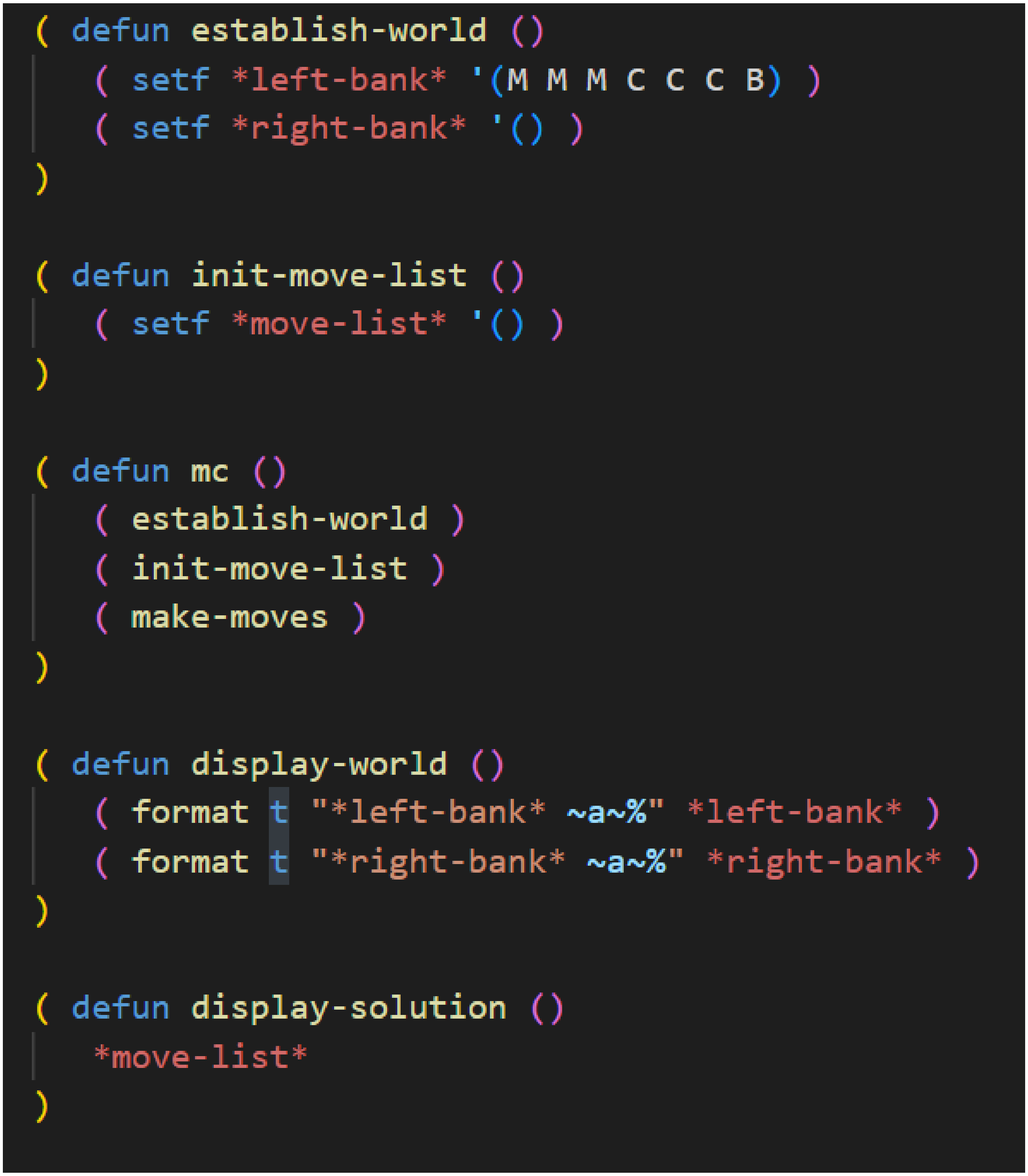
Programming Challenge: M&C Interactive Problem Solver

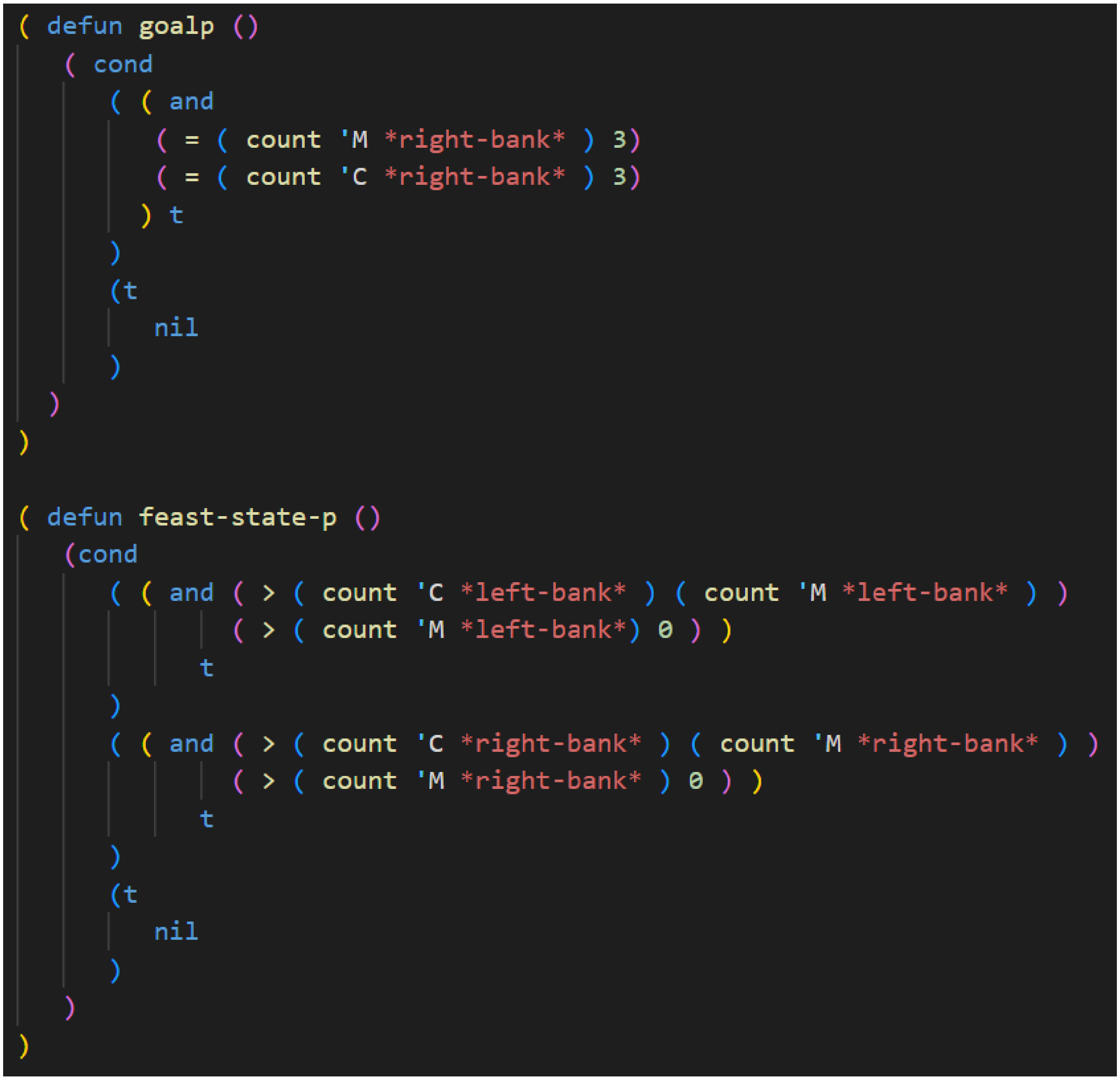
Abstract:

This program emulates the classic Missionaries and Cannibals logic puzzle. It is a nice way to work through possible solutions and will reveal an inapplicable or fatal move. It even has an option to reveal the move list used to achieve a viable solution (or not viable if you want to review where a mistake was made).

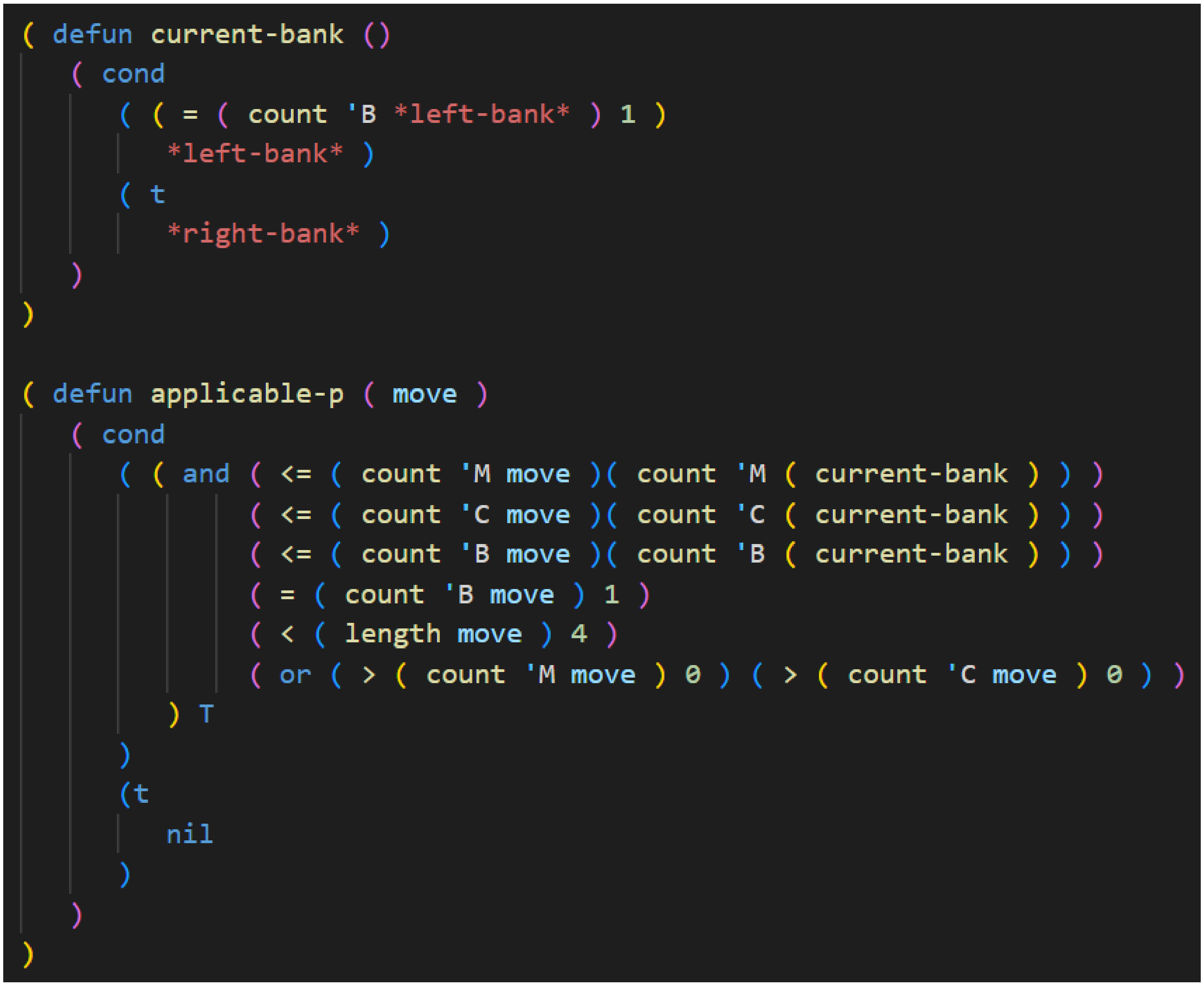
MC Code



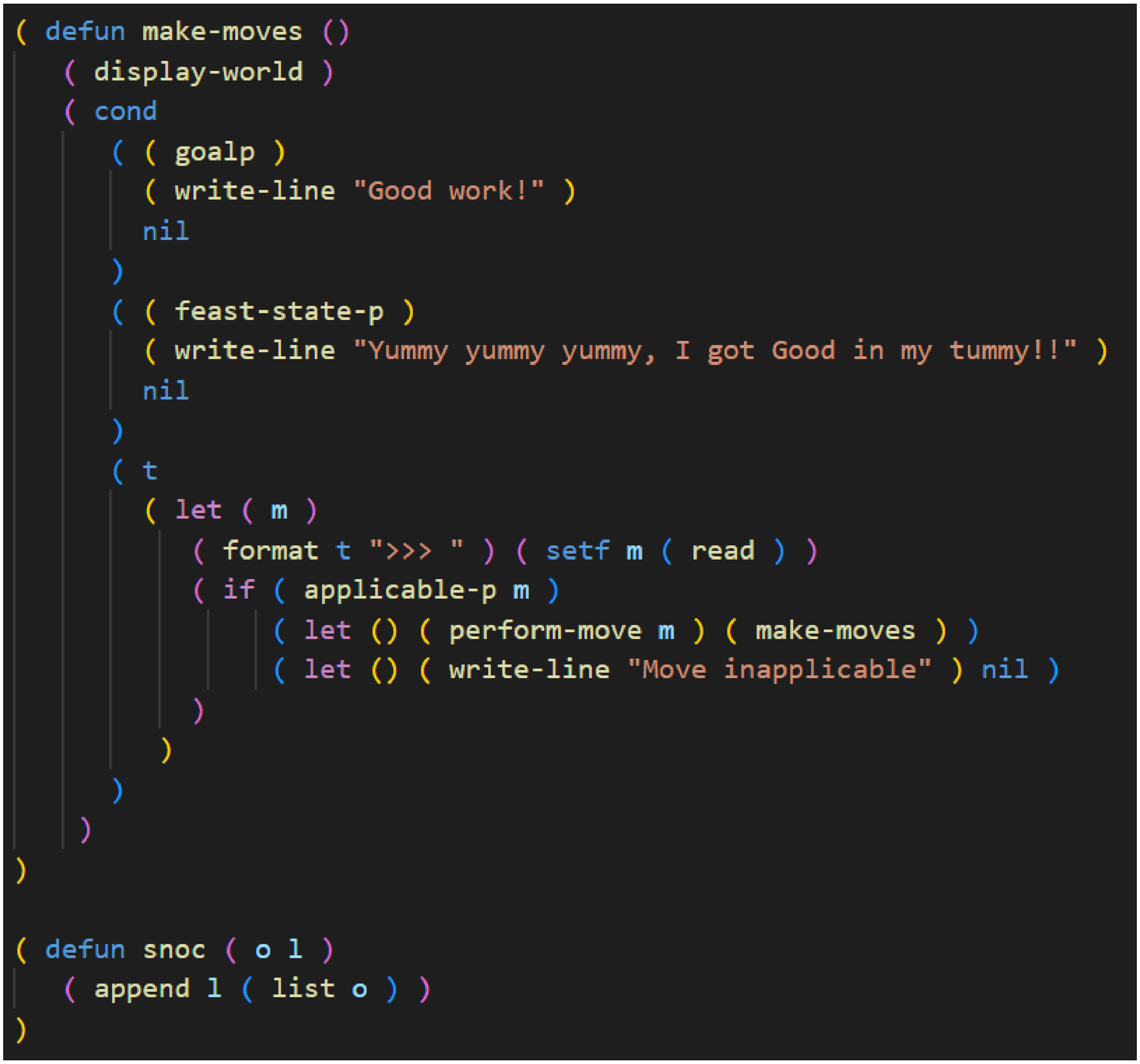
```
( defun establish-world ()
   ( setf *left-bank* '(M M M C C C B) )
   ( setf *right-bank* '() )
)

( defun init-move-list ()
   ( setf *move-list* '() )
)

( defun mc ()
   ( establish-world )
   ( init-move-list )
   ( make-moves )
)

( defun display-world ()
   ( format t "*left-bank* ~a~%" *left-bank* )
   ( format t "*right-bank* ~a~%" *right-bank* )
)

( defun display-solution ()
   *move-list*
)
```



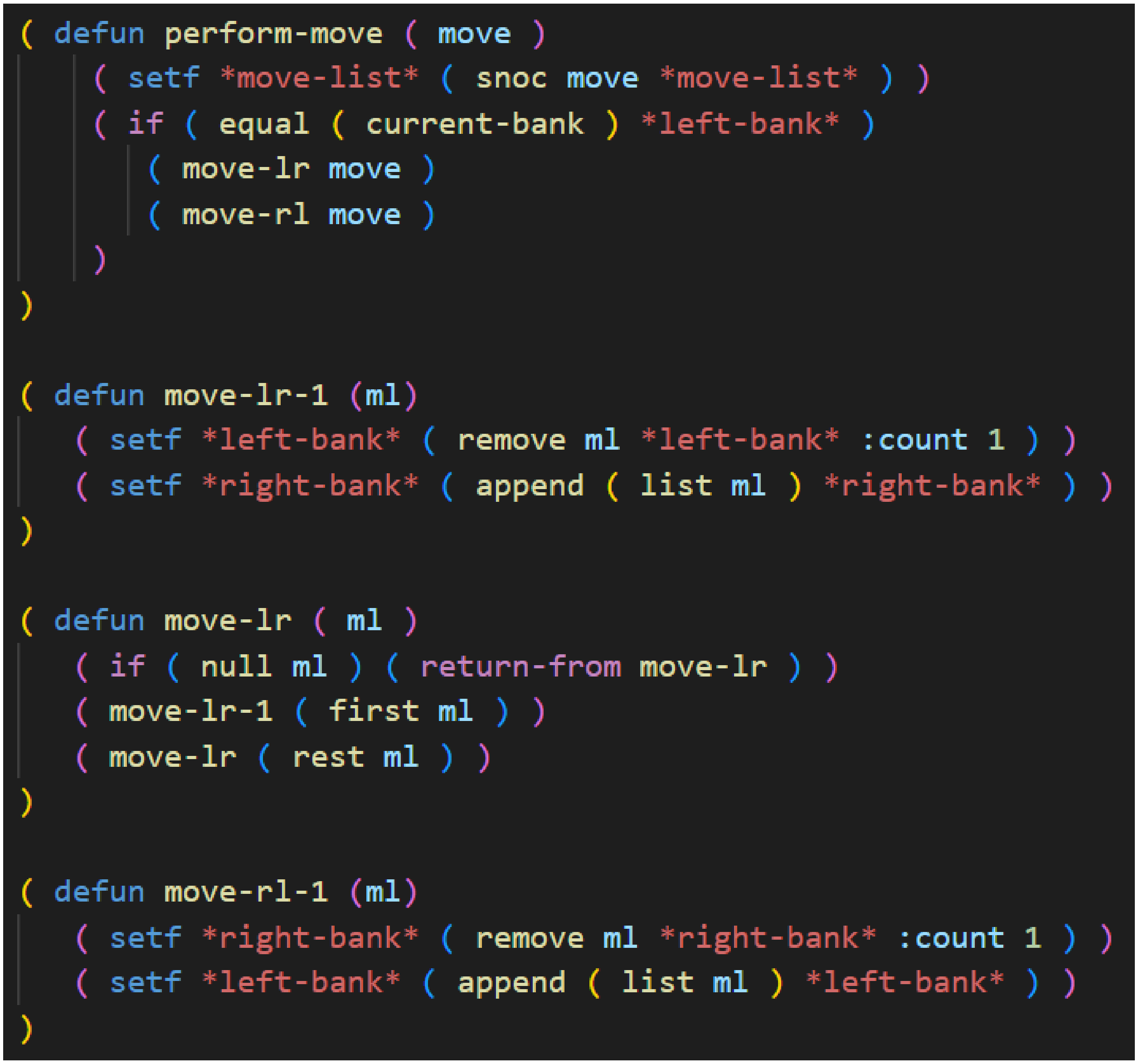
```
( defun goalp ()
   ( cond
      ( ( and
         ( = ( count 'M *right-bank* ) 3)
         ( = ( count 'C *right-bank* ) 3)
        ) t
      )
      (t
         nil
      )
  )
)

( defun feast-state-p ()
   (cond
      ( ( and ( > ( count 'C *left-bank* ) ( count 'M *left-bank* ) )
              ( > ( count 'M *left-bank*) 0 ) )
            t
      )
      ( ( and ( > ( count 'C *right-bank* ) ( count 'M *right-bank* ) )
              ( > ( count 'M *right-bank* ) 0 ) )
            t
      )
      (t
         nil
      )
   )
)
```



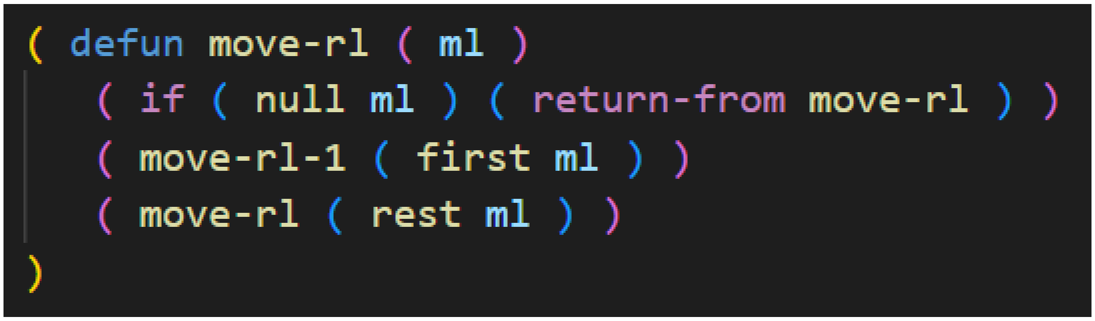
```
( defun current-bank ()
   ( cond
      ( ( = ( count 'B *left-bank* ) 1 )
         *left-bank* )
      ( t
         *right-bank* )
   )
)

( defun applicable-p ( move )
   ( cond
      ( ( and ( <= ( count 'M move )( count 'M ( current-bank ) ) )
              ( <= ( count 'C move )( count 'C ( current-bank ) ) )
              ( <= ( count 'B move )( count 'B ( current-bank ) ) )
              ( = ( count 'B move ) 1 )
              ( < ( length move ) 4 )
              ( or ( > ( count 'M move ) 0 ) ( > ( count 'C move ) 0 ) )
         ) T
      )
      (t
         nil
      )
   )
)
```



```
( defun make-moves ()
   ( display-world )
   ( cond
      ( ( goalp )
        ( write-line "Good work!" )
        nil
      )
      ( ( feast-state-p )
        ( write-line "Yummy yummy yummy, I got Good in my tummy!!" )
        nil
      )
      ( t
        ( let ( m )
           ( format t ">>> " ) ( setf m ( read ) )
           ( if ( applicable-p m )
                ( let () ( perform-move m ) ( make-moves ) )
                ( let () ( write-line "Move inapplicable" ) nil )
           )
         )
      )
    )
)

( defun snoc ( o l )
   ( append l ( list o ) )
)
```

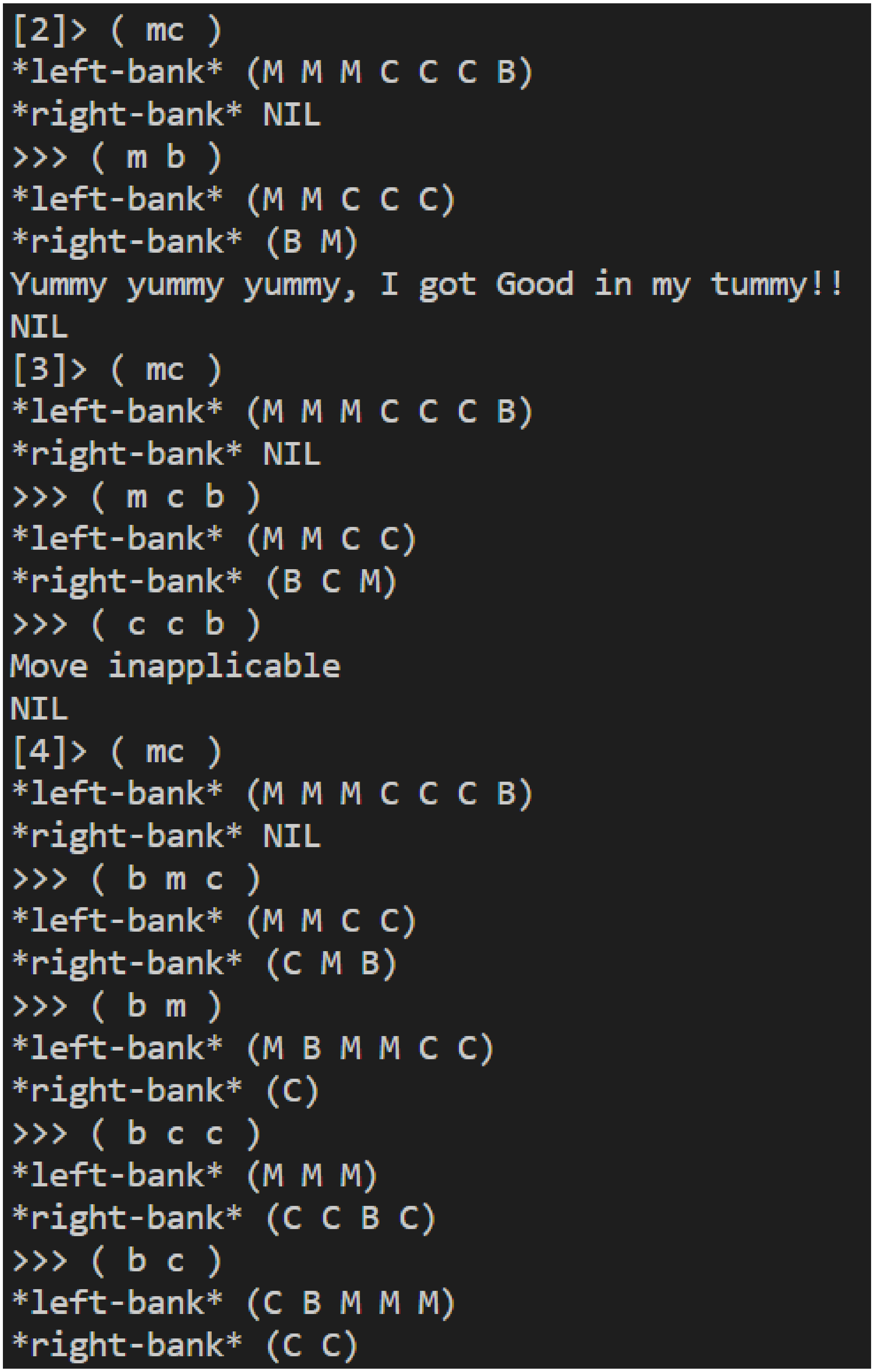


```
( defun perform-move ( move )
    ( setf *move-list* ( snoc move *move-list* ) )
    ( if ( equal ( current-bank ) *left-bank* )
        ( move-lr move )
        ( move-rl move )
    )
)

( defun move-lr-1 (ml)
   ( setf *left-bank* ( remove ml *left-bank* :count 1 ) )
   ( setf *right-bank* ( append ( list ml ) *right-bank* ) )
)

( defun move-lr ( ml )
   ( if ( null ml ) ( return-from move-lr ) )
   ( move-lr-1 ( first ml ) )
   ( move-lr ( rest ml ) )
)

( defun move-rl-1 (ml)
   ( setf *right-bank* ( remove ml *right-bank* :count 1 ) )
   ( setf *left-bank* ( append ( list ml ) *left-bank* ) )
)
```

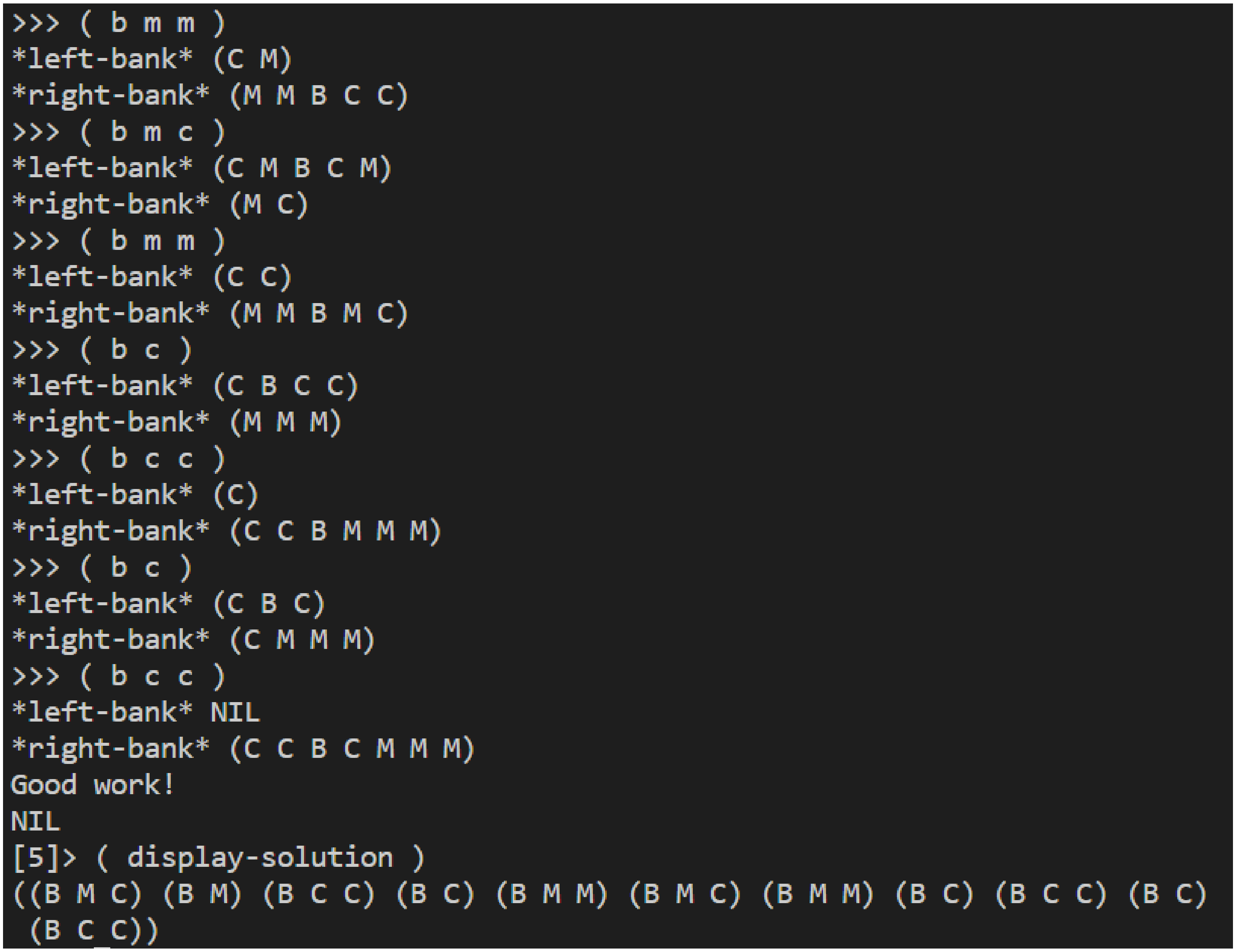


```
( defun move-rl ( ml )
   ( if ( null ml ) ( return-from move-rl ) )
   ( move-rl-1 ( first ml ) )
   ( move-rl ( rest ml ) )
)
```

MC Demo



```
[2]> ( mc )
*left-bank* (M M M C C C B)
*right-bank* NIL
>>> ( m b )
*left-bank* (M M C C C)
*right-bank* (B M)
Yummy yummy yummy, I got Good in my tummy!!
NIL
[3]> ( mc )
*left-bank* (M M M C C C B)
*right-bank* NIL
>>> ( m c b )
*left-bank* (M M C C)
*right-bank* (B C M)
>>> ( c c b )
Move inapplicable
NIL
[4]> ( mc )
*left-bank* (M M M C C C B)
*right-bank* NIL
>>> ( b m c )
*left-bank* (M M C C)
*right-bank* (C M B)
>>> ( b m )
*left-bank* (M B M M C C)
*right-bank* (C)
>>> ( b c c )
*left-bank* (M M M)
*right-bank* (C C B C)
>>> ( b c )
*left-bank* (C B M M M)
*right-bank* (C C)
```



```
>>> ( b m m )
*left-bank* (C M)
*right-bank* (M M B C C)
>>> ( b m c )
*left-bank* (C M B C M)
*right-bank* (M C)
>>> ( b m m )
*left-bank* (C C)
*right-bank* (M M B M C)
>>> ( b c )
*left-bank* (C B C C)
*right-bank* (M M M)
>>> ( b c c )
*left-bank* (C)
*right-bank* (C C B M M M)
>>> ( b c )
*left-bank* (C B C)
*right-bank* (C M M M)
>>> ( b c c )
*left-bank* NIL
*right-bank* (C C B C M M M)
Good work!
NIL
[5]> ( display-solution )
((B M C) (B M) (B C C) (B C) (B M M) (B M C) (B M M) (B C) (B C C) (B C)
 (B C C))
```